DXB211 Creative Coding Assignment 1

Design and Aesthetic Choices

The sketches I completed mostly revolved around shapes and collision. This is because I thought they offered the most creative outcomes whist maintaining simplicity. This simplicity was vital because it allows a larger audience to understand the sketches and comprehend how to interact with them appropriately (Glinert, 2008). For example, in the brief 1 sketch 1 I completed, I made sure to keep it very simple with the only interaction being through buttons and each button would create a unique shape on the canvas. Furthermore, in brief 2 sketch 1, brief 2 sketch 2 and brief 5 sketch 1 I limited the user's interaction to mouse pressed events only. Even though the interaction scope was limited, the feedback from the sketches, specifically after valid user interaction, was extremely noticeable through drastic changes. These changes included saturation, hue, size and even the introduction of new elements in the case of brief 2 sketch. Additionally, I utilized vibrant colours throughout all my sketches to increase engagement (Martinez, Adi and Prima, 2012).

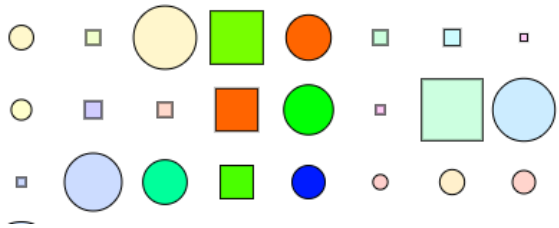

Figure 1
Here you can see that the middle elements have changed hue and saturation because they were previously clicked

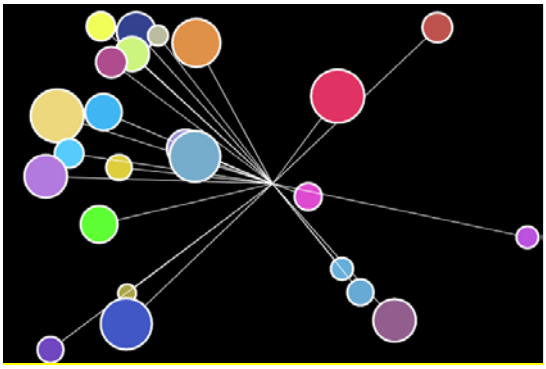

Figure 2
The sketch initially spawns only with 15 balls, however after the user initiates a `mousePressed()` event, a new ball spawns.

Figure 3
If a ball collides with the text, it causes the text to change colour. To change the colour manually, the user can trigger a `mousePressed()` event.

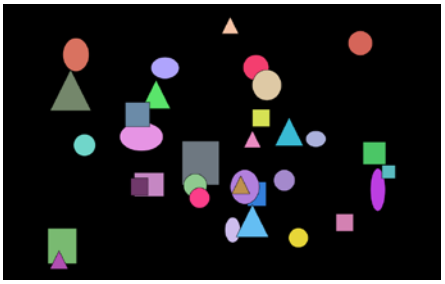

Figure 4
A snippet from Brief 1 Sketch 1 which showcases my initial utilization of vibrant, highly saturated, colours.

Approach to the Creative Process

At the start of the project I found out that I had to change my approach very quickly. This was due to the fact that I'm a heavily experienced programmer in combination with the fact that the sketches had to showcase an evident progression of knowledge as each week progressed. For example, I initially created Brief 2 Sketch 1 and Brief 2 Sketch 2 using object classes. Because this implementation was too advanced for the knowledge gained from the tutorials at the time, I had to refactor the code from object classes to simple variables and arrays. The basis of my approach revolved around keeping the primary `setup()` and `draw()` functions as minimal as possible by implementing other functions. This approach allowed for easy debugging, through strategic `print()` statement placement, and readability. Furthermore, through a majority of my sketches, I implemented makeshift 'objects' through the use of a unique index in multiple arrays.

```
let particleX = [];
let particleY = [];
let particleSize = [];
let particleVelocityX = [];
let particleVelocityY = [];
let particleColour = [];
```

Figure 5.1

To access a singular particle's variables, the index would be consistent throughout all the arrays. E.g. Particle 0's variables could be accessed through `particleX[0]`, `particleY[0]`, `particleSize[0]`, `particleVelocityX[0]`, `particleVelocityY[0]` and `particleColour[0]`.

```
function setup(){

  createCanvas(1200, 600);
  background(0);
  textSize(myFontSize);
  textFont(neonFont);
  randColour = colours[int(random(0, 5))];
  myTextWidth = textWidth("NEON");
  //This stores all the objects which contain the x and y position of the pixels which create the font
  myPoints = neonFont.textToPoints("NEON", (width - myTextWidth) / 2, height / 2, myFontSize);
  //Create a random number of particles
  numParticles = random(30, 50);
  //Create all the variables for all the particles
  for(let i = 0; i < numParticles; i++){
    createParticle(i);
  }
}

function draw(){

  background(0);
  //flooring the milliseconds ensures we've got a whole number
  let ms = floor(millis());
  fill(color(randColour));
  text("NEON", (width - myTextWidth) / 2, height / 2);

  for(let i = 0; i < numParticles; i++){
    //Change the colour of the lines that connect the particles every 15 milliseconds
    if(ms % 15 === 0){
      randLineColour = colours[int(random(0, 5))];
    }
    drawParticleLines(i, randLineColour);
    moveParticle(i);
    particleFontCollision(i);
    drawParticle(i);
  }
}
```

Figure 5.2

Implementing functions such as `createParticle()`, `drawParticleLines()`, `moveParticle()`, `particleFontCollision()` and `drawParticle()` allowed me to keep the `setup()` and `draw()` functions as clean as possible.

Development Log

Brief 1 Sketch 1

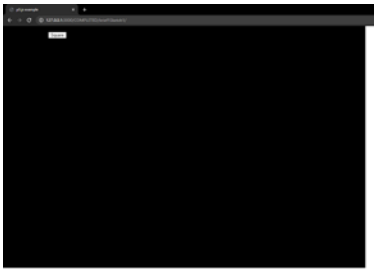

Figure 6.1
Implemented the first button.

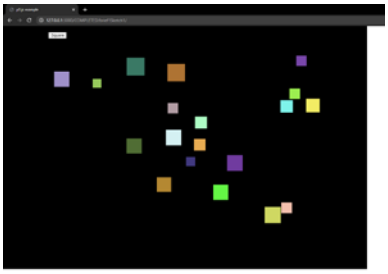

Figure 6.2
Implemented the `drawSquare()` function.

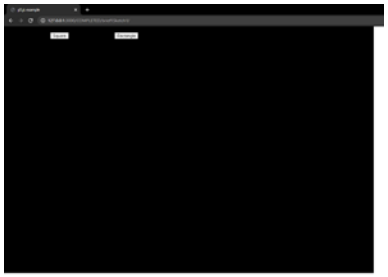

Figure 6.3
Implemented the rectangle button.

Figure 6.4
Implemented the `drawRect()` function.

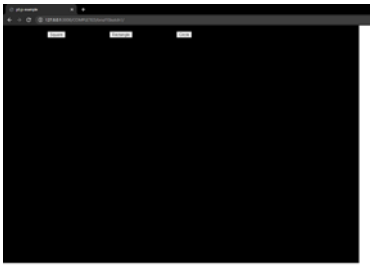

Figure 6.5
Implemented the circle button.

Figure 6.6
Implemented the `drawCircle()` function

Figure 6.7
Implemented the ellipse button.

Figure 6.8
Implemented the `drawEllipse()` button.

Figure 6.9
Implemented the triangle button.

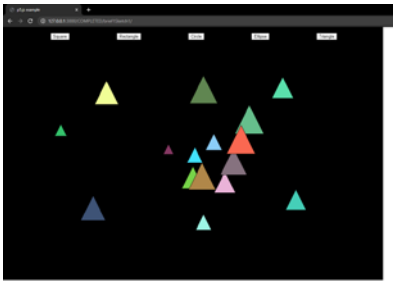

Figure 7.0
Implemented the `drawTriangle()` function.

Figure 7.1
Implemented the clear button.

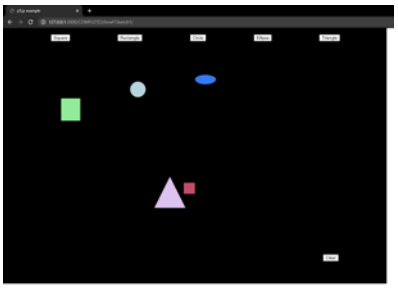

Figure 7.2
Image of the functions all successfully working after pressing each.

Figure 7.3
Implemented the `clear()` function.

Brief 2 Sketch 1

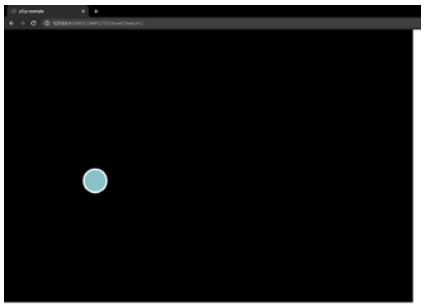

Figure 8.1
Implemented 1 ball.

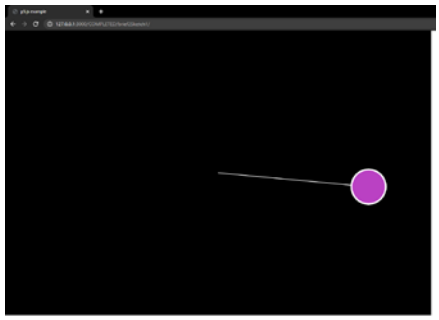

Figure 8.2
Drew a line from the ball to the centre of the canvas.

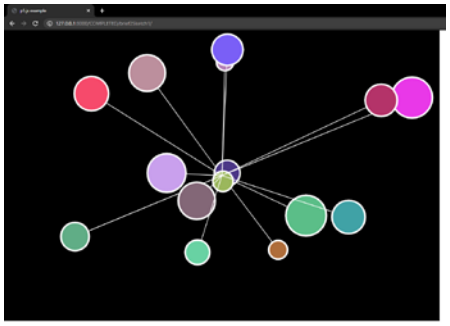

Figure 8.3
Drew multiple balls.

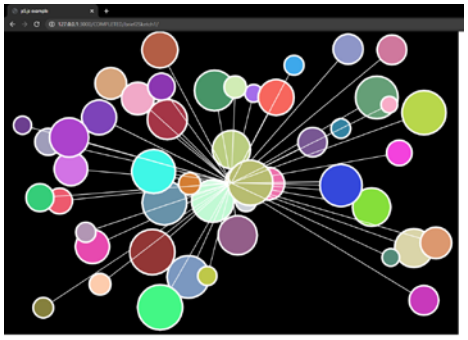

Figure 8.4
Implemented the ability to spawn a new ball when the mouse is pressed.

Brief 2 Sketch 2

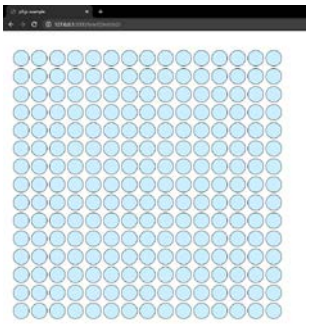

Figure 9.1
Implemented a nested for loop to draw all the circles.

Figure 9.2
Implemented a `random()` function to set a random size.

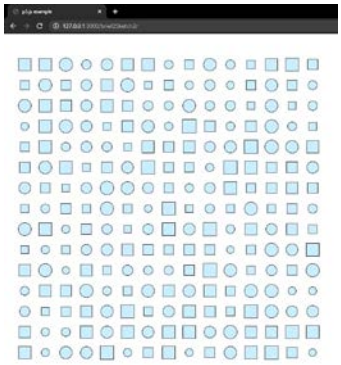

Figure 9.3
Implemented another `random()` function to determine whether to draw a circle or a square.

Figure 9.4
Implemented another `random()` function to get a random hue for the shape.

Figure 9.5
Implemented the ability for the shapes to grow in size progressively and shrink back down once they reach a predetermined size.

Figure 9.6
Implemented a regenerate button, which when pressed, respawns all of the shapes with a random type, size and colour.

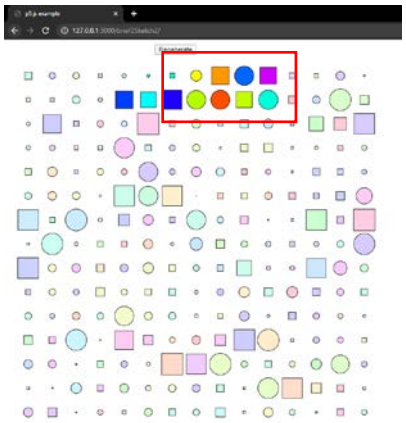

Figure 9.7
Implemented the ability for the user to shrink and change the shapes hue and saturation if they press them with a `mousePressed()` event. In the vacinity of the red rectangle, you can easily determine which shapes were pressed because of their drastically different hue and saturation values.

Figure 10.0
Positioned simple text in the middle of the canvas with a green hue.

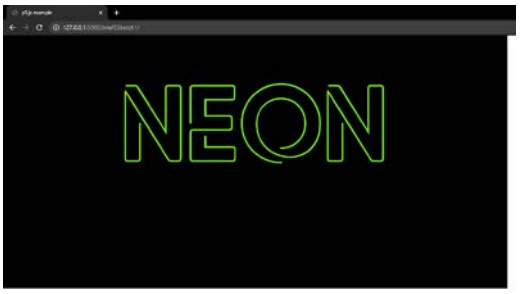

Figure 10.1
Downloaded and imported a custom ‘neonistic’ font.

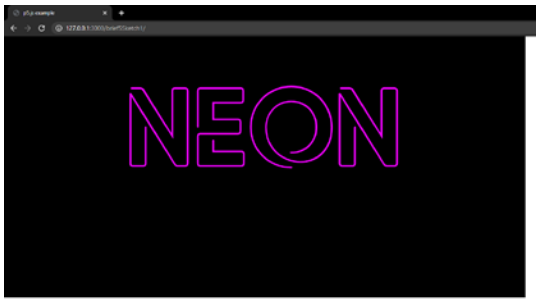

Figure 10.2
Set the hue of the text to change to a predefined random ‘neon’ colour after a `mousePressed()` event.

Figure 10.3
Spawned random particles in random positions around the canvas.

Figure 10.4
Gave the particles a random predefined ‘neon’ colour.

Figure 10.5
Implemented collision between the particles and the ‘neon’ text. Following collision, the ‘neon’ text would change hue and the respective particle would change trajectory.

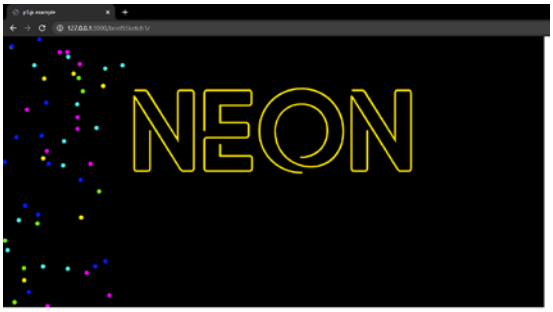

Figure 10.6
Sometimes the particles would spawn inside the text, therefore they couldn't escape due to the collision implementation. As a result, the x spawn position of the balls was constrained.

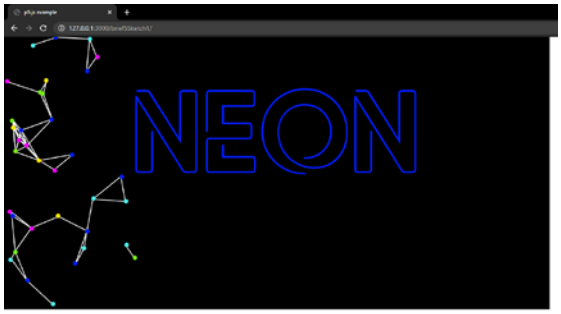

Figure 10.7
Drew lines from each ball, connecting each other, if they came within a certain distance.

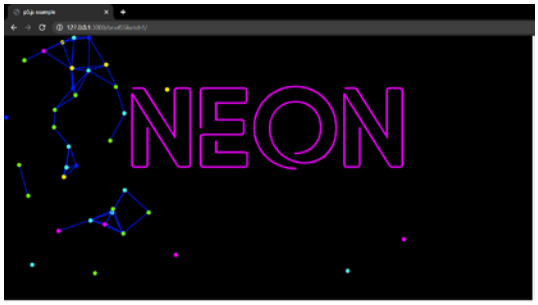

Figure 10.8
Implemented the ability for the connecting lines to change colour to a random 'neon' hue whenever the milliseconds passed, since the sketch was initially loaded, mod 15 was equivilent to 0.

Creative Influences

My Brief 2 Sketch 1 and Brief 5 Sketch 1 sketches were heavily influenced by the game ‘Pong.’ Pong was originally released in 1971 by game developer and publisher Atari on arcade cabinets. The physics behind the ball’s movement in ‘Pong,’ was implemented in the previously mentioned sketches. These sketches featured collision components which allowed the balls to bounce off the boundaries of the canvas with a random trajectory much like how the ball bounces off the paddles from ‘Pong’.

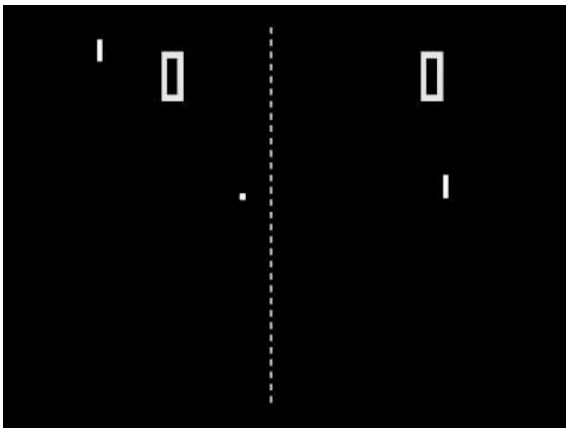

Figure 11
An image of the original 1971 release of the Atari game ‘Pong.’

My Brief 1 Sketch 1 and my Brief 2 Sketch 2 sketches were inspired by the baby toy titled ‘Shape & Sorted It Out.’ It’s a simple game which features the 3 basic shapes, a triangle, a square and a circle. I wanted to take the simplicity from this, in particularly the shapes, and expand on it in an interactive digital manner.

Figure 12
An image of the classic ‘Shape & Sort It Out’ toy.

The colour scheme for my sketches were heavily inspired by the phenomenon known as a ‘aurora.’ The hue and saturation of the light which appears in the sky during this phenomenon can be comparable to neon signs. I found the pink, green and blue hue present in this phenomenon to contrast well against each other. Additionally, I’ve previously worked with a pink, green and blue colour scheme in web design.

Figure 13
An image of the phenomenon known as a ‘aurora.’

References

Martinez, J. (2012). A Study of Colour as an Attribute that Intensifies User's Engagement in Game Plays. The International Journal Of Multimedia & Its Applications, 4(2), 1-20. doi: 10.5121/ijma.2012.4201. Accessed 22/04/2020. Available at URL: https://pdfs.semanticscholar.org/8e8f/acdf8eb4403f53db4fa5a7473ed593c9af10.pdf

Glinert, E. (2008). The human controller : usability and accessibility in video game interfaces. Retrieved 22 April 2020, from https://dspace.mit.edu/handle/1721.1/46106

My brief 2 sketch was inspired by the following example:

https://p5js.org/examples/motion-bouncy-bubbles.html

My brief 5 sketch 1 was inspired by the following example:

https://p5js.org/examples/simulate-particles.html

Collision logic was gathered from the P5.js collide library available at:

https://github.com/bmoren/p5.collide2D

Images

https://www.amazon.com/Plan-Toy-Shape-Sort-Out/dp/B00000ITIV

https://tkwsdmc.com/blog/europe-travel-best-places-to-see-the-northern-lights/